Parallel Merging and Sorting on Linked List

Yijie Han
School of Computing and Engineering
University of Missouri at Kansas City
Kansas City, MO 64110
Email: hanyij [AT] umkc.edu

Sreevalli Tata
School of Computing and Engineering
University of Missouri at Kansas City
Kansas City, MO 64110
Email: st8d7 [AT] mail.umkc.edu

Abstract – **We study linked list sorting and merging on the PRAM (Parallel Random Access Machine) model. In this paper we show that n real numbers can be sorted into a linked list in constant time with $n^{2+\varepsilon}$ processors or in $O(\log\log n)$ time with n^2 processors. We also show that two sorted linked lists of n integers in {0, 1, …, m} can be merged into one sorted linked list in $O(\log^{(c)} n\sqrt{\log\log m})$ time using $n/(\log^{(c)} n\sqrt{\log\log m})$ processors, where c is an arbitrarily large constant.**

Keywords- Parallel algorithms, optimal algorithms, EREW(Exclusive Read Exclusive Write), CREW(Concurrent Read Exclusive Write), CRCW(Concurrent Read Concurrent Write).

I. INTRODUCTION

In this paper we study parallel merging and sorting. The computation models we use are the EREW (Exclusive Read Exclusive Write) PRAM (Parallel Random Access Machine), the CREW (Concurrent Read Exclusive Write) PRAM and the CRCW (Concurrent Read Concurrent Write) PRAM [1]. On a PRAM memory is shared among all processors. On the EREW PRAM in one step no more than one processor can read or write one memory cell. On the CREW PRAM multiple processors can read one memory cell in one step but no more than one processor can write into one memory cell in one step. On the CRCW PRAM multiple processors can read or write into one memory cell in one step. When multiple processors write into one memory cell in one step an arbitration scheme needs to be used to decide the result written into the memory cell. On the Priority CRCW PRAM the highest priority processor wins the write among the processors writing into the memory cell. The priority can be the index of the processor. On the Arbitrary CRCW PRAM an arbitrary processor wins the write. On the Common CRCW PRAM when multiple processors write the same memory cell in one step they have to write the same value and that value is writing into the memory cell. Among these three variants of CRCW PRAM, Priority CRCW is the strongest model, Arbitrary CRCW PRAM is weaker than the Priority CRCW PRAM, and Common CRCW PRAM is the weakest among the three. In this paper we will use the Common CRCW PRAM and the Arbitrary CRCW PRAM.

Let T_p be the time complexity of a parallel algorithm using p processors. Let T_1 be the time complexity of the best serial algorithm for the same problem. Then $pT_p \geq T_1$. When $pT_p=T_1$ then this parallel algorithm is an optimal parallel algorithm.

When we have a T_P time algorithm using $_P$ processors, then when we use p processors the time can be expressed or translated as $T_PP/p+T_P$.

A parallel algorithm for a problem of size n using polynomial number processors (i.e. n^c processors for a constant c) and running in polylog time (i.e. $O(\log^c n)$ time for a constant c) is regarded as belong to the NC class [2], where NC is Nick's class. Researchers in parallel algorithm field are working to achieve NC algorithms and optimal parallel algorithms.

In the conventional setting, the result of merging or sorting is placed in an array with small numbers precedes larger numbers. It is known that merging takes at least O(loglogn) time with n/loglogn processors on the CREW PRAM [3] and sorting takes at least O(logn) time [4] on the EREW PRAM and at least O(logn/loglogn) time [5] on the CRCW PRAM with polynomial number of processors.

In order to avoid these lower bounds for merging and sorting researchers studies other variants of merging and sorting. For example, if we do not require the sorting result of n numbers be placed in an array of size n we can do better than the O(logn/loglogn) time if we allow the sorted result be placed in an array of size larger than n with n memory cells storing the sorted data in ascending order and other memory cells filled with blank or letting the memory cells between two sorted number be filled with either one of these two numbers. This is called padded sorting and can be done in O(loglogn) time with n^2 processors [6].

We have been working on another line of merging and sorting, namely we consider placing the sorted result of merging and

sorting on a linked list with smaller numbers precede larger numbers on the linked list. We have previous experience achieving better complexity by sorting integers into a linked list [7]. In this paper we show that if we use $n^{2+\varepsilon}$ processors then we can sort n real numbers into a linked list in constant time. If we use n^2 processors then we can sort n real numbers into a linked list in O(loglogn) time. Another result we want to demonstrate is to merge two sorted linked lists (smaller numbers precede larger numbers on the linked list) into one sorted linked list in $O(\log^{(c)} n\sqrt{\log\log m})$ time using $n/(\log^{(c)} n\sqrt{\log\log m})$ processors, where $\log^{(1)}n=\log n$, $\log^{(i)}n=\log\log^{(i-1)}n$ and c is an arbitrarily large constant.

Previoulsy Bhatt et al [8] and Hagerup [9] showed that integers in {0, 1, …, m-1}can be sorted into linked list in O(nloglogm/p+loglogm) time using p processors. When p>n their algorithm can be improved to O(loglogm/log(p/n)+1) time. No previous algorithms are known to sort real numbers into a linked list except to sort them into an array in Ω(logn) time on the EREW PRAM or Ω(logn/loglogn) time on the CRCW PRAM. Here we sort them into a linked list in constant time.

Previous merging results are about merging two sorted arrays. These includes Kruskal's [10] and Valiant's [11] results of merging real numbers in O(n/p+loglogn) time using p processors on the CREW PRAM and Berkman and Vishkin's result [12] of merging integers in {0, 1, …,m-1} in O(n/p+logloglogm) time on the CRCW PRAM.

The main novelty of our results is that we noticed that real numbers can be sorted into a linked list much faster than sorting them into an array. Our merging algorithm is also applied to merging two sorted linked lists. Previous results deal with sorting and merging them in arrays.

II. SORT N REAL NUMBERS INTO A LINKED LIST

Let A [0...n-1] be the array of n input real numbers. We are going to sort these real numbers on a linked list in constant time using $n^{2+\varepsilon}$ processors, where ε is an arbitrary small positive constant, or sort them on a linked list in O(loglogn) time using n^2 processors. We do this on the Common CRCW PRAM.

First we find the minimum element m in A. This can be done in constant time with n^2 processors [10] on the Common CRCW PRAM. Call this element m. Let MIN=m-1.

Then for each element A[i] we will copy array A to a new array A_i. This takes constant time with n^2 processors. We then compare A[i] with every element $A_i[j]$ in A_i. If $A[i] < A_i[j]$ or $(A[i] == A_i[j]$ && i<=j) then we will do $A_i[j]$ =MIN. Then we will find the maximum element $A_i[k]$ in A_i. This takes constant time using $n^{1+\varepsilon}$ processors (or O(loglogn) time with n processors) for A_i [10]. For all i=0, 1… n-1, this takes constant time with $n^{2+\varepsilon}$ processors (or O(loglogn) time with n^2 processors). $A_i[k]$ is the largest element smaller than A [i]. Thus we can make a link from A [k] to A [i].

Thus we have described a Common CRCW PRAM algorithm that can sort n real numbers into a linked list in constant time with $n^{2+\varepsilon}$ processors, or in O(loglogn) time with n^2 processors.

Theorem 1: All the n real numbers can be sorted into a linked list in constant time with $n^{2+\varepsilon}$ processors or in O(loglogn) time with n^2 processors on the Common CRCW PRAM.

We have not been able to reduce the number of processors significantly while keeping the time. We are working on it to see whether it is possible to reach an optimal NC algorithm [2] with o(logn/loglogn) time.

III. MERGE TWO SORTED LINKED LIST OF N INTEGERS EACH INTO A SORTED LINKED LIST

It is known that n integers in {0, 1, 2…m-1} can be sorted into a linked list using nt processors in O(loglogm/logt) time on the Arbitrary CRCW PRAM [13, 14]. Here we show how to merge two sorted linked lists of n integer each into a sorted linked list in $O(\log^{(c)} n\sqrt{\log\log m})$ time using $n/(\log^{(c)} n\sqrt{\log\log m})$ processors, where $\log^{(1)}n=\log n$ and $\log^{(i)}n=\log\log^{(i-1)}n$, and c is an arbitrarily large constant.

It is known that two arrays of n sorted integers can be merged in O (n/p+logloglogm) time using p processors [12] on the Arbitrary CRCW PRAM. Here we provide an alternative as our algorithm is for merging two sorted linked lists.

If we have two arrays of sorted integers in array A and B, then we can sample every t-th integer in them, i.e. picking the A[0], A[t], A[2t] … and B[0], B[t], B[2t]… When sorted integers are on linked lists L and M we cannot sample every t-th integer directly. We will have to use linked list contraction. Linked list contraction can contract every t_1 consecutive integers (nodes) on a linked list into a super node, where $t \le t_1 \le 2t$. This can be done on the EREW PRAM in $O(n/p+\log t\log^{(c)} n)$ time for an arbitrary large constant c using p processors [15]. After nodes have been contracted into super nodes then we sample the first node (integer) in every super node. This will give us O(n/t) sampled integers when L and M each has n integers in the linked list.

We will let $t=\sqrt{\log\log m}\,2^{\sqrt{\log\log m}}$.

So, total time taken by contraction of a linked list is $O(n/p+\log^{(c)} n\log t) = O(n/p+\log^{(c)} n\sqrt{\log\log m})$ using p processors on the EREW PRAM.

We then sample the first node in each super node. This gives us $n/(\sqrt{\log\log m}\, 2^{\sqrt{\log\log m}})$ sampled node. We can use $2^{\sqrt{\log\log m}}$ processors for each sampled node (thus $n/\sqrt{\log\log m}$ processors for the linked list). Thus we can sort sampled nodes from both linked lists into a sorted linked list in $O(\log\log m/\log 2^{\sqrt{\log\log m}}) = O(\sqrt{\log\log m})$ time on the Arbitrary CRCW PRAM [13, 14]. When we have p processors this translates to time $O((n/\sqrt{\log\log m})\sqrt{\log\log m}/p+\log^{(c)} n\sqrt{\log\log m}) = O(n/p+\log^{(c)} n\sqrt{\log\log m})$ time.

After the sampled nodes merged into a linked list we have to insert integers that have not been sampled into the linked list. Let $l_1, l_2 \ldots l_t$ be the sampled integers from linked list L and let $m_1, m_2 \ldots m_t$ be the sampled integers from linked list M. When they have been sorted on a linked list they appear as $n_1, n_2 \ldots n_{2t}$. Now there are no more than t integers from L that are between n_i and n_{i+1} (i.e. larger than n_i and smaller than n_{i+1}). Because if there are at least t integers from L between n_i and n_{i+1} then there is at least 1 integer among them being sampled. However, between n_i and n_{i+1} there are no other integers being sampled.

Thus we need to merge the (no more than t) integers in L and the (no more than t) integers in M that are between n_i and n_{i+1}. Because these integers are contracted into super nodes and therefore the integers come from L can be placed in an array and the integers from M can be placed into another array. Thus we are talking about merging two arrays of t integers each into one array. This can be done in $O(t/p+\log\log t) = O(t/p+\log\log\log m)$ time [10,-11]. For all the input integers this becomes $O(n/p+\log\log\log m)$ time.

Therefore, overall our algorithm has $O(n/p+\log^{(c)} n\sqrt{\log\log m})$ time with p processors on the Arbitrary CRCW PRAM.

Note that the space used by our algorithm is not linear. This is because we need space to place the nodes in a super node. Because we need O(t) space for a super node to place the node contracted into it, thus the space requirement is $O(nt) = O(n\sqrt{\log\log m}\, 2^{\sqrt{\log\log m}})$. We can reduce the space to linear by merging nodes in two super nodes by accessing the binary tree for the nodes in the super node built during the linked list contraction. However, this may require more than $\sqrt{\log\log m}$ time because we cannot use indexing to access nodes.

Theorem 2: Two sorted linked lists of size n each can be merged into a sorted linked list in $O(n/p+\log^{(c)} n\sqrt{\log\log m})$ time using p processors on the Arbitrary CRCW PRAM.

IV. CONCLUSIONS

We studied sorting and merging on linked list. Our sorting algorithm allows us to sort n real numbers into a linked list in constant time with $n^{2+\varepsilon}$ processors or in O(loglogn) time with n^2 processors. The most intriguing part of this is that we have not been able to reduce We studied sorting and merging on linked list. Our sorting algorithm allows us to sort n real numbers into a linked list in constant time with $n^{2+\varepsilon}$ processors or in O(loglogn) time with n^2 processors. The most intriguing part of this is that we have not been able to reduce the number of processors significantly while keeping the time. Our linked list merging algorithm can merge two sorted linked lists of integers into one sorted linked list. Although this algorithm is slower than the O(logloglogm) time for merging two arrays, it provides an alternative as it is for merging two linked lists instead of two arrays.

REFERENCES

[1] R. M. Karp, V. Ramachandran, Parallel algorithms for shared-memory machines. In *Handbook of Theoretical Computer Science (Vol. A): Algorithms and Complexity*, J. van Leeuwen, Ed., New York, NY: Elsevier, 869-941(1991).

[2]. S. A. Cook. Towards a Complexity Theory of Synchronous Parallel Computation. *L' Enseignement Mathématique,* 27, 99-124(1981).

[3]. A. Borodin, J. E. Hopcroft. Routing, merging and sorting on parallel models of computation. *Proc. 1982 ACM Sypm. On Theory of Computing (STOC'1982),* 338-344(1982)

[4]. S. A. Cook, C. Dwork, R. Reischuk. Upper and lower time bounds for parallel random access machines without simultaneous writes. *SIAM J. Comput.* Vol. 15, No. 1, 87-97(1986).

[5]. P. Beame, J. Hastad. Optimal bounds for decision problems on the CRCW PRAM. *Proc. 1987 ACM Symp. On Theory of Computing (STOC'1987)*, 83-93(1987).

[6]. T. Goldberg, U. Zwick. Optimal deterministic approximate parallel prefix sums and their applications. *Proc. 3rd. Israel Symp. On Theory and Computing Systems*, 220-228(1995).

[7]. Y. Han, X. Shen. Parallel integer sorting is more efficient than parallel comparison sorting on exclusive write PRAMs. *SIAM J. Comput.* 31, 6, 1852-1878(2002).
[8]. P.C.P. Bhatt, K. Diks, T. Hagerup, V.C. Prasad, T. Radzik, S. Saxena. Improved deterministic parallel integer sorting. *Information and Computation*, **94**, 29-47(1991).
[9]. T. Hagerup. Towards optimal parallel bucket sorting. *Information and Computation*. **75**, 39-51(1987).
[10]. C. P. Kruskal. Searching, merging, and sorting in parallel computation. *IEEE Trans. Comput.*, C-32, 942-946(1983).
[11]. L. G. Valiant. Parallelism in comparison provlems. *SIAM J. on Computing*, Vol. 4. No. 3, 348-355(1975).
[12]. O. Berkman, U. Vishkin. On parallel integer merging. *Information and Computation*. 106, 266-285(1993).
[13]. N. Goyal. An Arbitrary CRCW PRAM Algorithm for Sorting Integers into the Linked List and Chaining on a Trie. *Master's Thesis. University of Missouri at Kansas City*. 2020.
[14] Y. Han, N. Goyal, H. Koganti. Sort Integers into a Linked List. *Computer and Information Science*. Vol. 13, No.1, 51-57(2020).
[15]. Y. Han. Uniform linked lists contraction. In arXiv.org with paper id 2002.05034.